一种利用模块内聚性的对象抽取方法*

周毓明 徐宝文

(东南大学计算机科学与工程系 南京 210096)

E-mail: bwxu@seu.edu.cn

摘要 引入子程序-类型关系图来表示程序中类型和子程序之间的关系,讨论了模块内聚性的几个度量准则,并分析了增删子程序对模块内聚度的影响.在此基础上,给出了基于模块内聚性的对象抽取算法.

关键词 对象,对象抽取,内聚性,紧密度,重叠度.

中图法分类号 TP311

用面向对象技术开发的的软件,程序理解容易,后期维护方便.但现存的许多软件的开发语言或是非面向对象语言,或尽管是面向对象的语言但开发时没有使用面向对象的技术,再加上有些软件没有良好的编程结构,导致软件后期维护十分困难.因此,如果能在这样的系统中抽取出子程序与数据之间的关系,尽可能地还原出系统设计人员头脑中的"对象",并将其转换成用面向对象技术开发的系统,则可提高系统的可靠性、可维护性和可重用性.

这项工作的关键是通过对程序行为的详细分析,抽取出相关的数据和子程序,组合成有实际意义的对象.尽管在这方面已有不少工作[1~4],如 Liu 和 Widle 提出的基于全局变量和基于类型的方法[1]、Panose 等人提出的基于接收器的方法[3],但这些方法只是将数据和子程序按类型或全局变量简单的分类,没有考虑模块的性质,抽取出的对象有可能与实际中的对象不符.Canfora 在 Liu 和 Widle 的方法的基础上提出了用子图的内部联结度的方法[2],但该方法实际上还是利用全局变量来分类的,忽略了简单类型与复杂类型在对象抽取过程中的作用不同这个事实.为克服这些不足,本文提出一种基于模块内聚性的对象抽取方法.

1 子程序-类型关系图

正如人们所熟知的,子程序包括过程和函数,过程常带有形参声明,函数有形参声明和返回值类型声明,有时,子程序体内还涉及到全局变量的处理,因此子程序与形参类型、返回值类型和全局变量的类型紧密相关,子程序之间因这些类型而关联.在对象识别过程中,各类型的作用是不一样的,复杂类型比简单类型的作用大.为刻画类型的复杂程度,下面先引入类型复杂度的概念.如果类型 t_1 被用来定义类型 t_2,则称 t_1 是 t_2 的一个成分类型.类型 t 的复杂程度 $C_{t,l}$ 与它的所有成分类型的复杂程度和成分类型相对于 t 的嵌套层次数 l 有关(t 相对于本身的层次数是 0),其定义如下:

$$C_{t,l}=\begin{cases} v+l & \text{当 } t \text{ 为基本类型} \\ Complex * C_{t',l+1} & \text{当 } t \text{ 为指针类型 } (Complex>1), \\ f(C_{t_1,l+1},C_{t_2,l+1},\ldots,C_{t_n,l+1}) & \text{当 } t \text{ 为复合类型} \end{cases}$$

其中 v 为基本类型的复杂度,t' 为指针所指向的类型,$Complex$ 为指针类型的复杂程度.f 表示类型 t 的复杂程度是其所有成分类型 $t_1,t_2,\ldots,t_n$ 的复杂度 $C_{t_1,l+1},C_{t_2,l+1},\ldots,C_{t_n,l+1}$ 的函数,它可随具体的语言而定.例如,在 Ada

* 本文研究得到江苏省青蓝工程跨世纪人才基金资助.作者周毓明,1974 年生,博士生,主要研究领域为软件工程.徐宝文,1961 年生,教授,博士生导师,主要研究领域为程序设计语言,软件工程,并行与网络软件技术.

本文通讯联系人:徐宝文,南京 210096,东南大学计算机科学与工程系

本文 1998-09-14 收到原稿,1999-04-13 收到修改稿

中,当 t 为记录类型时,f 可取 $\sum C_{t_i,t+1}$(t_i 为记录的成分类型);当 t 为数组类型时,f 可取 $dim * 2 * C_{t',t+1}$(因为数组类型 t 的复杂性与它的维数 d 密切相关,但与数组元素具体数目没有太大关系.t' 为数组元素类型,这里乘 2 的原因是一维数组的复杂度显然比只有一个成分的记录复杂,经考虑乘 2 较合适).在此基础上有如下定义:

定义(子程序-类型关系图).令 S_{subp} 表示系统中的子程序集,S_{type} 表示系统中的类型集.如果图 $G=(N,E)$ 中 $N=S_{subp}\cup S_{type}$,$E=\{(p,t)\mid p\in S_{subp}\wedge t\in S_{type}\wedge t$ 为 p 中涉及到的类型$\}$,任一条边 (p,t) 上的权值为 $C_{t,0}$,则称图 G 为子程序-类型关系图.

对图 G 中的任一子程序结点 p,记其涉及的类型集为 $S_{p-t}(p)$,即 $S_{p-t}(p)=\{t\mid t\in S_{type}\wedge(p,t)\in E\}$;记与任一类型结点 t 有关的子程序集为 $S_{t-p}(t)$,即 $S_{t-p}(t)=\{p\mid p\in S_{subp}\wedge(p,t)\in E\}$.

这样,系统中子程序与类型之间的关系就体现在子程序-类型关系图上,最简单的对象识别方法就是在子程序-类型图上找出一个个独立的子图,每个独立的子图就构成一个对象.如果按这种方法抽取对象,有可能识别出的对象没有实际意义.这主要有两种情况.一种是将逻辑上属于不同对象的子程序封装在同一个对象中.如,栈对象有 Push_Stack (S: **in out** STACK_TYPE;Elem: **in** ELEM_TYPE)子程序,队列对象有 Enter_Queue (Q: **in out** QUEUE_TYPE; Elem: **in** ELEM_TYPE)子程序,在子程序-类型关系图上,这两个子程序因涉及到共同的类型 ELEM_TYPE 而相连,按上面的方法它们就封装在同一个对象中.另一种是将逻辑上属于不同对象的属性封装在同一个对象中.如,子程序 Init (S: **in out** STACK_TYPE; Q: **in out** QUEUE_TYPE)完成栈对象和队列对象的初始化,栈和队列在子程序-类型关系图上因子程序 Init 而在同一个子图中,按上面的方法就将栈和队列封装在同一个对象中.在这些情况下抽取出的对象显然没有实际意义.如果利用模块的内聚性来抽取对象,则可以避免上述无实际意义的对象的出现.为此,下面对模块的内聚性加以分析.

2 模块内聚性分析

模块由类型集以及操纵这些类型的子程序集组成,模块内子程序之间因涉及到对相同的类型处理而存在关联关系.模块的内聚性可用模块内部元素之间结合的紧密程度来刻画,从系统中抽取出的模块应符合软件工程高内聚、低耦合的原则,即抽取出的模块应具有较高的独立性,联系密切的元素不应分布在多个模块中,而应划分到同一个模块中.

对子程序-类型关系图 $G=(N,E)$,令 $MT_{int}(G)$ 表示图 G 中所有子程序共同涉及到的类型,即

$$MT_{int}(G)=\bigcap_{p\in S_{subp}} S_{p-t}(p).$$

$MT_{int}(G)$ 描述了模块内所有子程序之间涉及到的共同类型集.在下文中,为方便起见,用 $|S|$ 表示任何种类的集合 S 中的元素个数.为了描述表示模块的内聚度,下面引入模块紧密度、重叠度和内联度的概念.

模块 G 的紧密度 $Tightness(G)$ 用模块内所有子程序共同涉及的类型 $MT_{int}(G)$ 的相对复杂性之和与模块涉及的所有类型 S_{type} 的相对复杂性之和的比值来表示.模块 G 有一个高紧密度意味着模块 G 内各子程序涉及的共同类型比较多或涉及到的类型的复杂度比较高,即这些子程序紧密相关,这表明模块的内聚程度较高.亦即

$$Tightness(G)=\frac{\sum_{t\in MT_{int}(G)}(C_{t,0}*|S_{t-p}(t)|)}{\sum_{t\in S_{type}}(C_{t,0}*|S_{t-p}(t)|)}.$$

显然,当 $MT_{int}(G)=S_{type}$(即模块内所有子程序涉及的类型都相同)时,模块的结合度达到最大值 1.

模块 G 的重叠度 $Overlap(G)$ 表示在平均一个子程序涉及的类型中,$MT_{int}(G)$ 所占的复杂性比值,即在一个子程序中对公共部分处理的平均程度.重叠度高表明子程序与模块的大多数类型有关,从某方面反映了子程序间的高度关联关系,这也是模块具有高内聚的表现.由此,

$$Overlap(G)=(1/|S_{subp}|)\sum_{p\in S_{subp}}\frac{\sum_{t\in MT_{int}(G)}C_{t,0}}{\sum_{t\in S_{p-t}(p)}C_{t,0}}.$$

Canfora 在他的对象抽取方法中,为变量引用图中的子程序提出了子程序内部联结度的概念,然后根据各子程序的内部联结度对变量引用图作相应的合并、切片和删除,最后得到许多独立的子图,每个独立子图都视为一

个候选对象.但他的对象抽取方法仅与子程序处理的变量个数有关,而与变量的复杂程度无关,这显然会带来一些问题.由于我们使用的是子程序-类型关系图,且考虑到子程序之间的关联程度不仅与它们处理的类型个数有关,而且与类型的复杂程度密切相关,因此,必须对 Canfora 的子程序的内部联结度重新定义.

我们将子图的内部联结度定义为两个顶点都在子图内的所有边上的权值之和与至少有一个顶点在子图内的所有边上的权值之和的比值.这里,边上的权值用该边一端的类型顶点的复杂度计算.子程序 p 的内部联结度是指由涉及的类型是 $S_{p-t}(p)$ 子集的子程序集与类型集 $S_{p-t}(p)$ 组成的子图的内部联结度,因此,我们得到子图内部联结度 $IC(p)$ 的形式定义如下:

$$IC(p)=\frac{\sum'_{p'\in P(p)}\sum'_{t\in S_{p-t}(p')}C_{t,0}}{\sum_{t\in S_{p-t}(p)}(C_{t,0}*|S_{t-b}(t)|)},$$

其中 $P(p)=\{p_i|S_{p-t}(pp_i)\subseteq S_{p-t}(p)\}$.

模块的紧密度、重叠度和子程序内联度是模块内聚性的反映,它们的值越大,表明模块的内聚度越高.有了这些度量手段,用下一节的算法就可以在子程序-类型关系图中找出这样的一些子图:它们是由类型结点和子程序结点组成的子图,具有较高的内聚性,并且在一定程度上可以避免将逻辑上属于不同对象的子程序封装在同一个对象内,或将逻辑上属于不同对象的属性封装在同一个对象中.

3 对象抽取方法

3.1 增加和删除子程序对模块内聚度的影响

由于在子程序-类型关系图上,抽取过程中子图需要进行合并和分裂,这时需判断向模块增删子程序对模块内聚度的影响,为此,下面我们先定性分析向模块增加和删除子程序给内聚度造成的影响.设添加或删除的子程序名为 p',其涉及的类型为 $S_{p-t}(p')$,改变后的模块记为 G'.

3.1.1 向模块内增添一个子程序对模块内聚度的影响

当向模块 G 内添加一个子程序 p' 时,由模块紧密度、重叠度及 MT_{int} 的定义可直接推得以下两个引理.

引理 1.

$$\frac{\sum_{t\in MT_{int}(G)\cap S_{p-t}(p')}C_{t,0}-\sum_{t\in MT_{int}(G)-(MT_{int}(G)\cap S_{p-t}(p'))}(C_{t,0}*|S_{p-t}(p')|)}{\sum_{t\in S_{p-t}(p')}C_{t,0}}\geqslant Tightness(G)\Leftrightarrow$$

$$Tightness(G')\geqslant Tightness(G).$$

引理 2.

$$\frac{\sum_{t\in MT_{int}(G')}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}\geqslant(|S_{subp}|+1)*Overlap(G)-\sum_{p\in S_{subp}}\frac{\sum_{t\in MT_{int}(G')}C_{t,0}}{\sum_{t\in S_{p-t}(p)}C_{t,0}}\Leftrightarrow Overlap(G')\geqslant Overlap(G).$$

引理 1 表明,在模块所共同涉及的类型的复杂度增值与新增子程序涉及的类型复杂度之和的比大于等于原模块的紧密度时,新模块的紧密度才不会变小.引理 2 表明,模块重叠度的变化取决于新增子程序涉及的新模块中的公共类型的复杂度之和与其涉及的所有类型复杂度之和的比值与原模块重叠度之间的关系.而由模块内联度定义,显然,当 $S_{p-t}(p')\cap S_{p-t}(p)=\varnothing$ 时,内联度不变;当 $S_{p-t}(p')\subset S_{p-t}(p)$ 时,内联度变大;在其他情况下,内联度变小.

由上面的分析易知,当新添子程序涉及的类型包含了原模块涉及的公共类型时,如果新增子程序涉及的公共类型的相对复杂性与其涉及的总类型复杂性之比大于原模块的紧密度或重叠度,则模块的内聚度会保持不变或增大.即有如下推论.

推论 1. 当 $MT_{int}(G)\subseteq S_{p-t}(p')$ 时,

① 若 $Overlap(G)\leqslant\frac{\sum_{t\in MT_{int}(G)}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}$,则 $Overlap(G')\geqslant Overlap(G)$;

② 若 $Tightness(G)\leqslant\frac{\sum_{t\in MT_{int}(G)}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}$,则 $Tightness(G')\geqslant Tightness(G)$.

特别地,当 $MT_{int}(G)=S_{p-t}(p')$时,$Overlap(G')\geqslant Overlap(G)$,$Tightness(G')\geqslant Tightness(G)$.

3.1.2 从模块内删除一个子程序对模块内聚度的影响

删除子程序与增添子程序的过程正好相反,当从模块内删除一个子程序 p'时,由模块紧密度、重叠度及 MT_{int}的定义可直接推得以下两个引理.

引理 3.

$$\frac{\sum_{t\in MT_{int}(G)}C_{t,t}-\sum_{t\in MT_{int}(G')-MT_{int}(G)}(C_{t,0}*|S_{t-p}(t)|)}{\sum_{t\in S_{p-t}(p')}C_{t,0}}\leqslant Tightness(G)\Leftrightarrow Tightness(G')\geqslant Tightness(G).$$

引理 4.

$$\frac{\sum_{t\subset MT_{int}(G')}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}\leqslant\sum_{p\in S_{subp}}\frac{\sum_{t\in MT_{int}(G')}C_{t,0}}{\sum_{t\in S_{p-t}(p)}C_{t,0}}-(|S_{subp}|-1)*Overlap(G)\Leftrightarrow$$

$$Overlap(G')\geqslant Overlap(G),$$

子程序 p 内联度的变化则取决于被删子程序 p'涉及的类型与 p 涉及的类型之间的关系.当 $S_{p-t}(p')\cap S_{p-t}(p)=\varnothing$时,内联度不变;当 $S_{p-t}(p')\subseteq S_{p-t}(p)$时,内联度变小;在其他情况下,内联度会变大.

由引理 3 可知,模块紧密度的变化取决于被删子程序造成的模块所共同涉及的类型复杂度的减少量和它所涉及类型的复杂度的比值与原模块紧密度之间的关系.引理 4 则说明,模块的重叠度可能增大也可能减小,它取决于被删子程序涉及的新模块中的公共类型的复杂度之和与其涉及的所有类型复杂度之和的比值与原模块重叠度之间的函数关系.易知,当删除子程序 p'后,模块涉及的公共类型没改变时,如果被删子程序 p'涉及的原模块公共类型的相对复杂性与其涉及的总类型复杂性之比小于原模块的紧密度或重叠度时,则模块的内聚度会保持不变或增大.即有如下推论.

推论 2. 当 $MT_{int}(G)=MT_{int}(G')$时,

① 若 $Overlap(G)\geqslant\frac{\sum_{t\in MT_{int}(G)}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}$,则 $Overlap(G')\geqslant Overlap(G)$;

② 若 $Tightness(G)\geqslant\frac{\sum_{t\in MT_{int}(G)}C_{t,0}}{\sum_{t\in S_{p-t}(p')}C_{t,0}}$,则 $Tightness(G')\geqslant Tightness(G)$.

上面讨论了怎样提高一个候选模块的内聚度的充分条件,定性地分析模块的内聚度和模块的改变对内聚度的影响,从而为候选模块的抽取提供了合并与分裂的原则.下面,我们在此基础上讨论对象抽取算法.

3.2 对象抽取算法

子图之间存在一致性连接(coincidental connection)和假连接(spurious connection)两种联系方式[2].一致性连接指的是一个子程序因实现多个功能而存取多个数据结构,每个功能逻辑地属于一个对象.这时,可将子程序分割成多个子程序,每个子程序实现一种功能,逻辑地属于一个对象.如,子程序 Init 完成栈和队列的初始化,这时可将 Init 分割成两个不同的子程序,分别实现对栈和队列的初始化.而假连接是子程序实现特定的功能而存取不同的数据结构.如,子程序 Copy 将栈中的元素复制到队列中,这时将其分割没有意义,所以在处理中将该子程序移去.对子程序的程序流图进行分析,就可判断出该子程序是一致性连接还是假连接.

在子程序-类型图上,每个类型可得出对应的一个子图,计算这些子图的内聚度.由于抽取出的模块要具有较高的内聚度,所以只考虑那些内聚度大于阈值的子图.如果这些子图的子程序结点集的交集不为空,则应根据增加或删除子程序对这些子图内聚度的影响程度来判定该子程序划归哪一个子图.如果子图的类型结点集的交集不为空,则应判定是否应将这些子图合并.对象抽取算法如下:

(1) 在子程序-类型关系图中,对每个类型 t,记由子程序集 $S_{t-p}(t)$和类型集 $\bigcup_{p\in S_{t-p}(t)}S_{p-t}(p)$组成的子图为 M_t,计算 $Tightness(M_t)$或 $Overlap(M_t)$(由上面内聚度的分析可知,其值越大,表示该子图的内聚度越高),若 M_t 的紧密度或重叠度大于一给定阈值 $Step$,则将 M_t 加入 OverStep 列表.

(2) 对任意的子图 $M_{t_1},M_{t_2}\in OverStep$,当两子图的类型结点的交集中包含类型 t_1,t_2 时,分如下几种情况处理:

① 如果 M_{t_1},M_{t_2}合并为一个新的子图 M_t 后,子图的内聚度大于阈值 $Step$,则表明这两个类型是同一个对象中两个联系密切的重要属性,因此将合并后的子图 M_t 加入 OverStep 列表.重新生成子程序-类型关系图,在这个新的类型关系图上,类型结点 t_1,t_2 合并为一个新类型结点 t,在原图上,所有从子程序结点指向类型 t_1,t_2 的弧在新图上都指向类型结点 t.

② 如果 M_{t_1},M_{t_2}合并为一个新的子图 M_t 后,子图的内聚度小于阈值 $Step$.这时,如果是某个子程序同时存取 t_1,t_2 而造成两子图 M_{t_1},M_{t_2}之间的假连接,则将这个子程序从 M_{t_1}和 M_{t_2}中同时去掉;如果是某个子程序造成的一致性连接,则将其分割成几个逻辑功能独立的子程序.

(3) 重复(2),直到对任意 $M_{t_1},M_{t_2}\in OverStep$,这两个子图的类型结点的交集中不同时包含类型 t_1,t_2 为止.

(4) 对任意两个子图 $M_{t_1},M_{t_2}\in OverStep$,设它们子程序结点的交集为 Set.我们认为每个子程序在逻辑上只属于一个对象,因此,Set 中的子程序应划分给不同的子图.记 M_{t_1},M_{t_2}中去掉 Set 中子程序结点及相关联的边后的子图分别为 M'_{t_1},M'_{t_2}.对 Set 中的任意子程序 p,如果 p 划归 M'_{t_1}或 M'_{t_2}都导致其内聚度小于 $Step$,则抛弃 p.否则,如果有下面 3 种情况之一出现:

① p 划归 M'_{t_1}后其内聚度增加而划入 M'_{t_2}后其内聚度减小;

② p 划入 M'_{t_1}后内聚度增加的程度大于划入 M'_{t_2}后内聚度增加的程度;

③ p 划入 M'_{t_1}后内聚度减小的程度大于划入 M'_{t_2}后内聚度减小的程度,

则 p 应划归 M'_1.将改变后的子图加入 OverStep 列表.

(5) 重复(4),直到对任意 $M_{t_1},M_{t_2}\in OverStep$,这两个子图的子程序结点的交集为空为止.

(6) 将 OverStep 列表中的每个子图的类型结点作为属性,子程序结点作为方法封装成一个类.

算法中的阈值 Step 可通过对大量模块的内聚度统计分析后得到,也可作为一个参量来调节,以观察和分析不同内聚度层次对由系统中抽取出的对象的影响.

3.3 对象间继承关系的抽取

为便于抽取对象间的继承性,下面,我们非形式地引入一些基本概念.一个对象的类型用$[l_i:B_i^{i\in 1..n}]$表示,其中 l_i 是对象的方法名或属性名,B_i 是 l_i 的类型.若用 $A<:B$ 表示类型 A 是类型 B 的子类型,则蕴含是:如果 $A<:B$ 且 $a:A$,则 $a:B$.蕴含的含义是一个类型的子类型的实例也是该类型的实例.如果两个对象所对应的对象类型之间存在子类型关系,则这两个对象之间存在蕴含关系,即一个对象继承了另一个对象的属性和方法.对象继承性的抽取过程就是找出相应对象类型的子类型关系的过程.对象类型的子类型关系的不同定义对应于对象间继承关系的不同层次.下面引入一个较强的对象类型子类型的定义:如果子类型关系 $B_i<:B'_i,\forall i\in 1..n$ 成立,则有$[l_i:B_i^{i\in 1..n+m}]<:[l_i:B'^{i\in 1..n}_i]$.

抽取对象间的继承性分为下面几步:首先用上一节的算法在程序中抽取出对象,且形式化地表示出对象的类型;然后通过对象类型的演算找出对象类型间的子类型关系,抽取出对象间的继承关系;最后将对象类型及其继承关系转换为相应对象间的继承关系,因子对象继承父对象的属性和方法,所以转换过程中子对象的属性和方法可能要进行适当的增删.

4 结束语

从代码中抽取出抽象层次的对象,毕竟是从原程序出发,与设计人员头脑中的"对象"有一定的差距,况且程序本来就未必真正实施了设计人员真正需要的对象.但通过逆向工程的方法抽取出程序中的"拟对象",不仅有助于理解原系统的设计思想,而且能提高软件的可维护性和代码的可重用性.Liu 和 Widle 的基于全局变量和基

于类型的方法以及基于接收器的方法没有考虑模块的性质,只是将数据和子程序按类型或全局变量简单地加以分类,有可能将本来属于同一个对象的方法或属性分散到多个模块中,从而造成所取出的对象有可能与实际中的对象不符.Canfora 的方法在某些情况下得出的结果可能不理想.本文从模块的内聚性出发,将关联程度高的子程序、类型组合成对象,从而抽取出的对象内部具有较高的内聚度.我们已结合"Ada 逆向工程与软件维护支撑技术研究"课题开发出一个原型系统.它基本的功能是,从用 Ada83 开发的软件中识别并抽取出各种对象,并将其转换为 Ada95 程序,其中最重要的部分就是服务性任务到保护对象的转换[5].在将来的工作中,我们期望在对象抽取的实现中能加进语义知识的处理.

参考文献

1 Liu S S, Wilde N. Identifying objects in a conventional procedural language: an example of data design recovery. In: Proceedings of the IEEE Conference on Software Maintenance. San Diego, CA: IEEE Computer Society Press, 1990. 266~271

2 Canfora G, Cimitile A, Munro M *et al*. A reverse engineering method for identifying reusable abstract data types. In: Proceedings of the 1st IEEE Working Conference on Reverse Engineering. Baltimore, MD: IEEE Computer Society Press, 1993. 73~82

3 Livadas Panose, Johnson Theodore. A new approach to finding objects in programs. Journal of Software Maintenance: Research and Practice, 1994,6:249~260

4 Pedrycz Witold, Waletky James. Fuzzy clustering in software reusability. Software-Practice and Experience, 1997,27(3): 245~270

5 Li Bang-qing, Xu Bao-wen, Yu Hui-ming. Transforming Ada severing tasks into protected objects. In: Proceedings of the ACM SIGAda Annual International Conference. Washington, DC: ACM Press, 1998. 240~245

An Object-Extracting Approach Using Module Cohesion

ZHOU Yu-ming XU Bao-wen

(*Department of Computer Science and Engineering Southeast University Nanjing* 210096)

Abstract In this paper, an St (subprogram-type) graph is introduced to represent the relation between subprograms and types in programs. Several module cohesion metrics are discussed, and the effects of adding a subprogram to or deleting a subprogram from a module are analyzed based on module cohesion. An object-extracting algorithm is proposed.

Key words Object, object-extracting, cohesion, tightness, overlap.